Author's Original Manuscript Forthcoming in:
Journal of Management Information Systems, Special Section on the Transformative Value of Cloud Computing

KEY AFFORDANCES OF PLATFORM-AS-A-SERVICE: SELF-ORGANIZATION AND CONTINUOUS FEEDBACK

Oliver Krancher (corresponding author)
Institute of Information Systems, University of Bern, Engehaldenstr. 8, 3012 Bern, Switzerland
Email: oliver.krancher@iwi.unibe.ch, Phone: +41 31 631 4783

Pascal Luther†
Zuehlke Engineering AG, Bogenschützenstrasse 9A, 3008 Bern, Switzerland
Email: pascal.luther@zuehlke.com, Phone: +41 79 505 4461

Marc Jost†
ipt Innovation Process Technology AG, Marktgasse 28, 3011 Bern, Switzerland
Email: marc.jost@ipt.ch, Phone: +41 79 503 4456

†The second and the third author contributed equally.

Oliver Krancher is an assistant professor of information systems at the University of Bern, Switzerland. He holds a diploma from University of Regensburg and a Ph.D. from University of Bern. His research interests revolve around knowledge processes in the development, use, and management of information systems. His work has been accepted at outlets such as the Journal of the Association of Information Systems, the International Conference on Information Systems, and the European Conference on Information Systems. His teaching on cloud computing has been recognized with the AIS Innovation in Teaching Award. Prior to his academic career, he worked as a consultant in enterprise software implementation and outsourcing projects.

Pascal Luther is a business analyst at Zuehlke Engineering AG in Bern, Switzerland. He holds a master's degree in business administration with emphasis on information systems from the University of Bern. He also studied at Keio University, Japan and at Seoul National University, South Korea. His academic work on the adoption of platform-as-a-service in software development teams has been accepted for presentation at the International Conference on Information Systems. In this current position, he supports software development teams in agile software development methods and in requirements engineering.

Marc Jost is an information technology consultant at Innovation Process Technology AG in Bern, Switzerland. He holds a master's degree in business administration with emphasis on information systems from the University of Bern. His academic work focused on the interaction of technology and software development and its organizational impact. In his current occupation, he accompanies digital and agile transformation endeavors involving cloud computing, modern software development practices, and artificial intelligence.

ABSTRACT

Although software development teams increasingly use Platform-as-a-Service (PaaS), little is known about the impact of PaaS on software development. We explored the impact of PaaS on software development through a grounded-theory study, conducting 48 interviews in 16 teams. The data turned our attention to the affordances, or potentials for action, that PaaS provides to software development teams. Two key affordances emerging from our data analysis were self-organizing and triggering continuous feedback. Actualizing these affordances helped accelerate the collective learning processes that underlie software development, thus supporting software development teams in their quest for agility. Our emerging theory explains how, why, and when these affordances arise. The key contribution of our paper lies in unveiling how the use of cloud computing technology can transform technology-mediated collective learning activities by helping to remove barriers to rapid feedback. Our findings also imply that organizations can leverage PaaS to facilitate the transition to agile and continuous software development practices, in particular in conjunction with cross-functional team designs.

Keywords: Cloud computing, Platform-as-a-Service, software development, feedback, collective learning, affordances, agile software development, agility, continuous integration, continuous delivery, DevOps, lean software development

INTRODUCTION

Software development teams increasingly use Platform-as-a-Service (PaaS), a cloud computing technology that provides application environments as elastic, on-demand services [56]. The market for public PaaS services soared from 3.8$ billion in 2015 to 7.2$ billion in 2016 and is expected to further grow by 20% per year [31]. Many major technology companies, such as Amazon with its PaaS service Elastic Beanstalk, are now competing with established PaaS providers, such as Salesforce.com with its Heroku and Force.com services [78]. The primary users of these services are software development teams [18].

PaaS allows them to quickly set up application environments, to easily deploy code to these environments, and to instantly scale them [49]. According to a recent survey run by Cloud Foundry (an open-source PaaS initiative) [18], PaaS has a transformative impact on software development. The fraction of respondents that reported development cycles shorter than one week rose from 16% to 46% after the adoption of PaaS. Developer productivity also nearly tripled according to the survey results.

Despite such claims of transformative impact, the academic literature has paid relatively little attention to the impact of PaaS on software development. The existing PaaS literature has focused on the inner workings of PaaS services [1, 21], on the business models and ecosystems of PaaS providers [3, 22, 36], and on the PaaS characteristics desired by software developers [32, 33, 75]. The few articles that discuss impact are conceptual. They suggest that the use of PaaS makes deployment and testing activities more efficient and that it might enable end user computing [26, 41, 49, 81]. However, empirical evidence of these assertions and, more broadly, of the impact of PaaS on the work of software developers is lacking. This dearth of empirical evidence is unfortunate because it is difficult to theorize the impact of a technology without examining the practices through which the technology is used [60].

Evidence of the impact of PaaS on software development practices would be practically relevant because the use of PaaS could help teams in a transition process that many software development teams are currently undergoing. In their quest to rapidly deliver innovative software, many software development teams attempt to follow movements such as agile [29] and lean [62] software development, continuous integration and delivery [41], and DevOps [2]. Common to these movements is the aim to increase agility (i.e., the ability to rapidly create, react to, and learn from change [19]) by adopting practices based on self-organizing and frequent feedback [28, 73]. While these practices have attracted strong interest in the industry and in the literature [24, 28], relatively little is known about how these practices are affected by the infrastructure that underlies software development. Conceptual articles from the software development literature have recently argued that PaaS, with its on-demand self-service characteristic, is an

enabler for self-organizing and continuous feedback practices [2, 77]. However, empirical insights into how, why, and when PaaS enables these practices are rare. This has led scholars to call for research on the relationship of cloud computing and agile practices [81, p. 48].

In sum, although PaaS is diffusing at a rapid pace and although providers claim PaaS to have a transformative impact, we lack empirically founded knowledge about the impact of PaaS on software development. Given this limited knowledge, we performed a grounded-theory study [34]. We began with the broad aim of exploring the impact of PaaS on software development. As our informants kept talking about the actions that PaaS enabled, we decided to focus our study on the affordances of PaaS, i.e., on the potentials for action that PaaS provides to software development teams. Specifically, we explored two questions: *What affordances does PaaS provide to software development teams? How, why, and when do these affordances arise?* We explored these questions through 48 interviews conducted in 16 software development teams.

We found that PaaS provides four affordances: shaping environments, reusing software services, self-organizing, and triggering continuous feedback. Shaping environments and reusing software services are basic affordances that, after a brief learning process, arise from two capabilities of PaaS services: rapid elasticity and abstraction. In contrast, self-organizing and triggering continuous feedback are higher-level affordances. They arise after teams actualize basic affordances and thereby produce outcomes, such as self-contained tasks, that enable higher-level affordances, such as self-organizing. We also found that two work environment characteristics (functional sub-teams, architectural dependencies) inhibit the actualization of higher-level affordances because they undermine increased self-containment.

Our study makes three important contributions. First, it provides a perspective on the transformative impact of cloud technology by unveiling how the rapid-elasticity and abstraction capabilities of cloud technology enable self-organizing and continuous feedback, thereby allowing teams to accelerate collective

learning processes and enhance agility. Second, we contribute insights to the software development literature into how cloud technology can ease the transition to current software development practices. Third, we add to the affordance literature that affordances can depend on work environment characteristics. In the remainder of this paper, we review the literature to which we integrated our grounded findings, report the methods and results, and discuss implications.

BACKGROUND LITERATURE

Cloud Computing and Platform-as-a-Service

We rely on the definition of cloud computing by the US National Institute of Standards and Technology (NIST) [56]. Among the essential characteristics of cloud computing according to the NIST definition are on-demand self-service (i.e., resources are provisioned automatically, without requiring human interaction) and rapid elasticity (i.e., resources are rapidly scalable). Hence, although cloud computing is sometimes regarded as a type of outsourcing or contrasted with on-premises infrastructure, it is a model for organizing shared computing resources such that automatic, on-demand access to these resources is possible, irrespective of who owns them and whether they are located "on or off premises" [56, p. 2].

Although cloud computing comprises Software-as-a-Service (SaaS, cloud-based applications), Platform-as-a-Service (PaaS, cloud-based application environments), and Infrastructure-as-a-Service (IaaS, cloud-based hardware), most academic work focuses on SaaS [81], examining issues such as adoption [6], usage continuance [7], ecosystems [16], user desires [71], governance [69, 80], and shadow information technology (IT) (i.e., SaaS use without organizational approval) [35, 82]. A common theme in some of these studies is that cloud technology, with its essential characteristic of on-demand self-service, may shift power in organizations, giving end users greater control over the technology that they use in their work [35, 80, 82].

PaaS has received comparatively little attention in the literature [65, 81]. PaaS denotes cloud-based application environments [3, 5, 56]. Application environments are the configured resources on which running instances of applications rely, including hardware, operating systems, application and database servers, and configurable software components [41, p. 277]. Key PaaS features include instant set-up of application environments, instant deployment of code to these environments, and automatic scaling [21].

Analysts distinguish two broad categories of PaaS services: model-driven PaaS (mPaaS) and deployment PaaS (dPaaS) [3, 40]. *MPaaS* services, such as Force.com and Mendix, provide cloud-based application environments that include configurable software components. These components allow model-driven development, i.e., developing software by configuration, without necessarily writing code [75]. For instance, Force.com allows developing applications by defining data models, reports, and workflows. Some observers speculate that mPaaS might enable end user computing (i.e., application development by business users) [81]. *DPaaS* services, such as Heroku, Cloud Foundry, and Amazon Elastic Beanstalk, are at a lower level of abstraction because unlike mPaaS services, they do not include components for model-driven development [40]. Benefits of dPaaS services according to the literature include more efficient deployment due to automated system administration tasks and more efficient testing due to identical testing and production environments [26, 41, 49, 81]. Although PaaS aims at supporting software development, evidence of the impact of PaaS on software development practices is scarce.

Software Development

This section provides a brief, selective background on the software development literature. This background shall help link our findings to the literature [34], rather than serve as an a-priori theoretical lens.

Software Development as a Collective Learning Activity

Although the software development literature draws on a variety of theoretical perspectives, an increasingly popular perspective views software development as a complex *collective learning activity* [9, 25,

37, 72, 73]. In this collective learning activity, team members gradually acquire knowledge about requirements, technologies, and existing applications, about designs and code that address the requirements with particular technologies and applications, and about ways for integrating these elements into a coherent software [10, 27, 46, 76]. We define *learning* as the process of knowledge acquisition, *knowledge* as the capacity to act in a particular context [61], and a *software development team* as the collective of people (typically developers and business users) that share the goal of building software. The notion of collective learning activity can be visualized by comparing the effort needed to develop a software a first time to the (often fictitious) effort needed to develop the same software a second time. The second-time effort is typically only a fraction of the first-time effort due to the learning that accrued during the first go [10]. A collective learning perspective helps explain why many current software development movements advocate practices based on self-organizing and on frequent feedback, as we will argue next.

Self-organizing

Self-organizing refers to a process in which some form of order emerges from agents' spontaneous local actions and interactions [44]. It contrasts with processes in which order is imposed from outside. A software development team is self-organizing if the team "decides what to do, how and when to do it, and no one outside the [team] directs those activities explicitly" [73, p. 358]. Practices based on self-organizing are a hallmark of the agile software development movement, which values "individuals and interactions over [externally imposed] processes" [29] and which advocates uniting business users and developers in a self-organizing team [29]. These teams make decisions about requirements, solution designs, and the distribution of work [66]. Self-organizing is also a key idea behind DevOps, which advocates joint teams of developers and system administrators with no rigid separation of roles between the two [42].

From a collective learning perspective, practices based on self-organizing recognize that much of the knowledge in software projects emerges over time. Self-organizing allows teams to make decisions based on this emergent, local knowledge [38, 62]. In contrast, non-self-organizing teams would need to

rely on “someone else to name in advance the practices and conditions for every situation, [which] … obviously breaks down quickly” [39, p. 121] due the key role of cognitive limitations in software work [45].

Although many software development teams are attempting to transition towards self-organizing practices, these transitions are hampered by barriers related to resources and culture. Self-organizing benefits from teams having control over all *resources* relevant for their task [30, p. 51], including infrastructure and knowledge. Yet, as realized by the DevOps movement, software development teams often lack control over infrastructure and knowledge to manage infrastructure [42]. They may therefore depend on the external order imposed by infrastructure teams who dictate, for instance, technologies and deployment schedules. While the DevOps movement suggests removing this barrier by including system administrators into development teams and by eliminating the rigid separation of roles, empirical evidence shows that developers and operations often take their traditional division of labor for granted [63]. In such teams, the transition to self-organizing practices is a relatively slow process of *cultural change* [63].

Frequent Feedback

A second practice advocated by current software development movements is frequent feedback. *Feedback* denotes “information about actions returned to the source of the actions” [79]. Feedback includes internal feedback (i.e., feedback that actors directly perceive from monitoring the outcomes of their actions) and external feedback (i.e., feedback provided by another person) [11]. For instance, a developer obtains internal feedback when she deploys her code and recognizes problems after deployment. She obtains external feedback, for example, when a business user tests her code and informs her about the results. These examples indicate that feedback processes operate during many software development activities, such as compiling, deploying, testing, experimenting, and reviewing.

Current software development movements increasingly recognize that *frequent feedback* accelerates collective learning processes. In traditional, plan-based software development, months may elapse from

the moment a user specifies a requirement until the user sees the software in action, or from the moment a developer writes code until she tests the code in a production-like environment [41]. To use a metaphor, these users and developers resemble tennis players who practice their service but who can observe the position where the ball hit the ground only months after each swing. They can learn only slowly from the outcomes of their actions, which is unfortunate if the amount of required learning is large. As a remedy, agile methods advocate frequent feedback through time-boxed iterations, often of a duration of a few weeks [73], which are seen as "learning cycles" [66, p. xiii]. Lean methods acknowledge that such time-boxed rhythms may still delay feedback. They recommend further "increas[ing] the frequency of the feedback loops" [62, p. 38] because "[t]he shorter these cycles are, the more can be learned" [62, p. 14]. This principle is put into practice by the continuous integration and continuous delivery (CI/CD) movement [41]. A key principle of CI/CD is that "faster feedback" [41, p. 12] is possible with technology that allows small code changes to "trigger the feedback process" [41, p. 12]. Such technology automatically integrates changes into the shared code base and automatically tests and deploys them.

Although many teams desire to follow practices based on frequent or continuous feedback, at least three barriers slow down feedback in organizational realities. A first barrier are *wait times related to deployment*. In many organizations, deployment is performed by specialized infrastructure teams who deploy at pre-planned schedules, rather than on demand [42]. In such arrangements, days or weeks may elapse until code is deployed to and tested in testing environments. Although teams can reduce these wait times by automating deployment processes (i.e., by implementing CI/CD), setting up automated deployment also takes time and can be complex [15, 41]. Hence, it is difficult to obtain frequent feedback on software, in particular during the early weeks of a project—a time when ideas about requirements and designs are often vague and feedback would be particularly valuable. A second barrier are *too large tasks.* Frequent feedback can only be triggered if large tasks are broken down into smaller units, such that the completion of each small unit can trigger the feedback process [14, 58]. Yet some naturally large

tasks may be difficult to split up in smaller independent parts. In such situations, feedback is delayed until a large task unit is completed. A third barrier are *differences between testing and production environments*. Such differences delay feedback because problems specific to production environments will not be discovered until code is deployed to production. Given these challenges, research has described the transition to continuous feedback practices as the quest for the holy grail [4], a "hump of pain" [48, 67], or a process that requires high levels of top management support and months or years to complete [28]. How particular infrastructure may help overcome these challenges remains empirically unexplored.

Affordance

While our data collection and analysis unfolded, affordances turned out to be a useful concept for theorizing the impact of PaaS on software development. Affordances are potentials for goal-oriented action that objects (such as PaaS) offer to actors (such as software development teams) [68]. For instance, a mobile phone affords communicating [12], and an electronic health records system affords standardizing and coordinating [68]. Four key properties of the affordance lens are (1) the focus on actions, (2) the concern with how objects facilitate actions, (3) the idea that potentials for action emerge between actors and objects, and (4) the role of the actors' goals. Given its *focus on actions*, a key tenet of the affordance lens is "to focus … on what an actor could do with the object" [12, p. 4], such as communicating, standardizing, and coordinating. The focus on actions has invited many scholars to study practices (i.e., recurrent collective actions) and their change with an affordance lens [50, 51, 54, 68].

While affordance studies focus on actions, they are also concerned with the mechanisms of how *objects facilitate these actions*. For instance, Strong et al. [68] found that an electronic health records system (an object) allowed hospitals to standardize data (an action) because of the standard data entry forms provided by the system (technical capabilities of the object). Scholars appreciate this balanced interest in social action *and* materiality (i.e., objects) given that "materiality's role in organizational change remains under-theorized" [52, p. 159]. Importantly, the claim that an object (e.g., PaaS) facilitates an action (e.g.,

a software development practice) means that the object *makes the action possible or easier*, not that the object is a necessary condition for the action. For instance, while an electronic health record system affords standardizing, "standardizing can be done separately from implementing" a system [68, p. 76].

Although objects play important roles in affordance conceptualizations, affordances arise in the *relation of objects and actors* through three mechanisms. First, affordances may depend on actor capabilities [12]. For example, a mobile phone allows communicating only if the actors are capable of using their phones. Second, *higher-level affordances* depend on the *immediate concrete outcomes* produced by the actualization of *basic affordances* [68]. For instance, Strong et al. [68] found that the electronic health records system afforded coordinating patient care across sites (a higher-level affordance) only after hospital staff actualized the basic affordances of capturing patient data and ubiquitously accessing data anytime (basic affordances), which produced easy access to patient data (an immediate concrete outcome). Third, scholars have speculated that affordances depend on characteristics of the *work environment* [68, 74]. However, existing studies do "not really address, how non-technology related … mechanisms [such as work environment characteristics] interact with an IT-enabled change process" [74, p. 833].

Another property of the affordance lens is that actions are assumed to be *goal oriented*. It is therefore important to consider the goals (e.g., agility) of the actors in the particular social setting (e.g., software development teams) to explain what affordances these actors choose to actualize [68, p. 70]. In sum, an affordance lens explains actions and the resulting outcomes with the entanglement of technological capabilities, the actors' capabilities, the actors' choices to actualize the affordance given their goals in their social setting, and the changes in the social setting due to the actualization of basic affordances.

METHODS

We used the grounded-theory method (GTM) [13], relying on critical-realist assumptions [55, 57]. The GTM is appropriate for our study because the flexibility of the method makes it well suited for developing

theory about a phenomenon on which little is known [8, p. 6], such as PaaS-based software development. Cornerstones of our philosophical stance of critical realism are that reality objectively exists, that access to reality is limited, and that uncovering causal mechanisms is critical in science [55, 57].

Sampling

We iterated between sampling, data collection, and analysis. Our sampling logic followed the GTM principles of constant comparison and theoretical sampling [8, 13, 70]. We began our study in 2014 by searching for informants that were part of software development teams, used PaaS, and had experience in non-PaaS projects. We then iteratively expanded this sample through seven waves of data collection, intermingled with analysis. In line with the principle of constant comparison, we sampled to maximize our ability to make comparisons. Our final sample of 16 software development teams (see Table 1) includes teams that vary in a number of dimensions, specifically (1) the software development rationale (i.e., why and for whom the team developed software), (2) whether they used PaaS, (3) the PaaS product, (4) whether the PaaS provider was external, (5) team size, (6) project duration, (7) organization size, and (8) the dominant software development methods in their organizations.

In line with the principle of theoretical sampling, these dimensions emerged only during data analysis. For instance, after we noted how small service providers and start-ups leveraged PaaS to enact agile software development practices, we wondered whether we would observe similar practices in in-house units of large organizations with traditions in plan-based development. We therefore included five teams from such a company, Telco. Since our sample at that time included mostly small teams engaged in short projects, we added TelcoMedia, a team of over 30 people engaged in a multi-year project. Moreover, while we had initially envisioned two separate studies on dPaaS and mPaas, we noted that the themes were highly similar. We therefore combined the data into one study. In line with the principle of theoretical saturation, we terminated data collection after interviews only corroborated existing findings [13, p. 113]. Figure 1 illustrates saturation by plotting the cumulated number of codes against interviews.

Table 1: Overview of Software Development Teams

Team	*Software Development Rationale*	*PaaS Product*[a]	*PaaS Provider*[b]	*Team Task*	*Team Size*[c]	*Organization Size*[d]	*Org. SD Method*[e]
AppCo	Product company, service provider	Force.com (mPaaS)	External	Develops enterprise software package; offers configuration and extension of the packages to commercial customers	Small	Very small	Agile
BPMCo	Product company	CF (dPaaS)	External	Develops business process management software	Large	Small	Agile
CRMCo	Service provider	Force.com (mPaaS)	External	Offers custom development, often during or after implementing Salesforce.com for commercial customers	Small to medium	Small	Agile
DevCo	Service provider	CF, Heroku (dPaaS)	External	Offers custom development to commercial customers	Small to medium	Very small	Agile
ForceCo	Service provider	Force.com (mPaaS)	External	Offers custom development, often during or after implementing Salesforce.com for commercial customers	Small to large	Small	Agile
PaaSCo	Service provider	CF, Heroku (dPaaS)	External	Offers custom development to commercial customers	Small to medium	Very small	Agile
PayCo	Product company	CF, Heroku (dPaaS)	External	Develops payment services software to be used by private customers	Small	Very small	Agile
SecureCo	Internal software development	CF (dPaaS)	External	Develops embedded software for security systems	Medium	Large	Plan-based
ShareCo	Product company, Service provider	AppFog, Heroku (dPaaS)	External	Develops a resource sharing software package; offer configuration of the package to commercial customers	Small	Very small	Agile
SoloDev	Service provider	Heroku (dPaaS)	External	Offers custom development to commercial customers, heavily relying collaboration with freelancers	Small to medium	Very small	Agile
TelcoAPI	Service provider	CF (dPaaS)	Internal	Offers configuration and extension of services that make use of Telco's application programming interface	Small	Large	Plan-based
TelcoOrder	Internal software development	CF (dPaaS)	Internal	Develops order management software for internal customers	Small	Large	Plan-based
TelcoMedia	Internal software development	CF (dPaaS)	Internal	Develops a media storage software to be used by Telco's private customers	Large	Large	Plan-based
TelcoNetwork	Internal software development	CF (dPaaS)	Internal	Develops network information software for internal customers	Small	Large	Plan-based
TelcoNews	Internal software development	Non-cloud	-	Develops a news and collaboration platform to be used by internal customers	Medium	Large	Plan-based
TradCo	Service provider	Non-cloud	-	Offers custom development to external customers	Small	Small	Agile

[a] CF = Cloud Foundry, [b] PaaS provider: internal = PaaS provided by in-house unit, external = PaaS provided by external firm, [c] team size: small (<5 members), medium (5-15 members), or large (> 15 members), [d] organization size: very small (< 10 employees), small (10-100 employees), medium (101-10'000 employees), large (> 10'000 employees), [e] org. SD Method: dominant software development method in the organization

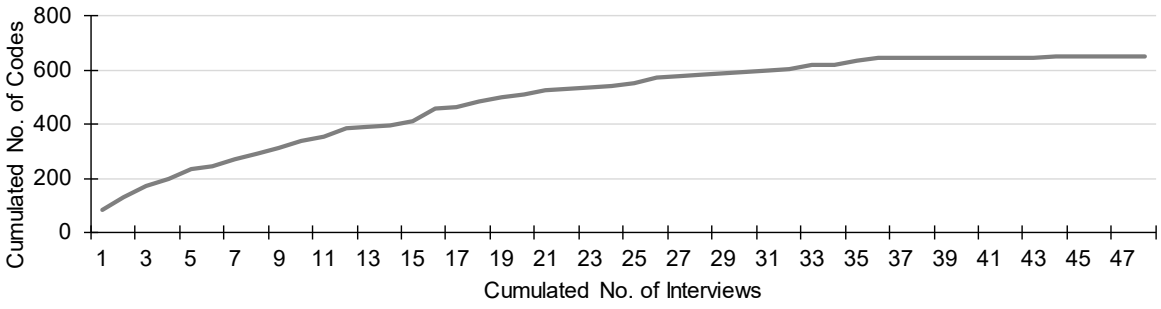

Figure 1: Saturation Plot

Data Collection

We conducted 48 interviews with 42 informants (see Table 2). The informants occupied a number of different roles, such as developers, owner-developers (i.e., individuals that owned part of the firm and developed software), architects, administrators, managers, testers, and external customers. 8 informants were interviewed twice to follow up on themes that emerged during data analysis and to explore whether their use of PaaS had changed over time. The average interview duration was 53 minutes. 40 Interviews were conducted face to face; 8 were conducted through videoconferencing. All interviews were audio-recorded and transcribed. Most of the quotes reported in this paper are translated from German.

As is often recommended for exploratory research, we used semi-structured interviews guided by a few open questions [20, p. 39]. This gave the informants ample opportunities to elaborate, and it allowed the interviewer to probe into emerging themes. In line with our initial goal to broadly explore the impact of PaaS on software development, the questions in the initial interview guide were: *(1) Please describe the software development project(s) you are working in. (2) How come you use PaaS? (3) How did the use of PaaS affect software development in this/these project(s) during different phases of the software development lifecycle? (4) How did the move to PaaS affect your team and collaboration within the team? (5) How important was the interaction with the PaaS provider?* Consistent with the GTM principle of no preconceived theoretical lenses [70], these questions did not focus on themes derived from a-priori theories but invited informants to talk about the change associated with the use of PaaS.

Table 2: Interviews

Team	*Informants*	*No. of interviews*
AppCo	2 owner-developers	2
BPMCo	1 owner-developer	2
CRMCo	3 developers (jointly present in 1 interview), 1 manager	2
DevCo	1 developer, 1 manager	2
ForceCo	3 developers, 3 managers, 2 external customers	9
PaaSCo	1 developer, 1 administrator, 1 manager	3
PayCo	1 owner-developer	2
SecureCo	2 developers	2
ShareCo	1 owner-developer	1
SoloDev	1 owner-developer	2
TelcoAPI	2 developers, 1 manager	4
TelcoOrder	1 developer, 1 manager	2
TelcoMedia	1 administrator, 1 architect, 1 manager, 1 performance engineer	5
TelcoNetwork	2 developers	3
TelcoNews	2 developers, 1 architect, 1 manager, 1 tester	5
TradCo	1 owner-developer, 1 manager	2

Data Analysis

Our data analysis followed the GTM procedures suggested by Charmaz [13]. We started with initial coding [13, p. 42], memo writing [13, p. 72], and visual mapping (i.e., visualizing events over time) [47], supported by NVivo. In line with the principle of no preconceived theoretical lenses [70], initial coding was not guided by the affordance lens or the notion of collective learning but was open to capture any theme related to the change of software development with PaaS. Initial coding resulted in many provisional codes that described the actions facilitated by PaaS, such as the example codes shown in Table 3. Initial coding produced many further themes that we followed during later data analysis, such as PaaS capabilities and work environment characteristics, as well as themes that we did not pursue further.

Table 3: Examples of initial codes

Code	*Example quotes*
Deploying frequently	“Deployment can be done more often much more easily."
Everybody can deploy	“Everybody can deploy.”
Starting on day one	“With PaaS, the development team can start right away, from day one.”
Trying out things	“I could very quickly try out Redis [a database technology] to see how it behaves and how fast it really is.”
Triggering immediate customer feedback	“We can show single features to customers within seconds. The customer can then look at the feature and provide his feedback very quickly.”

Initial coding was followed by focused coding, which centered on the provisional core category emerging

from our analysis: the potential for triggering continuous feedback offered by PaaS. We coded themes that described this potential, as well as themes that appeared related to it. A key strategy for uncovering these themes was constant comparison of data from PaaS-based software development to data from non-PaaS-based software development [13, 70]. For instance, the service providers DevCo, PaaSCo, and SoloDev used PaaS for some of the customers and non-PaaS technology for other customers. We compared the accounts of PaaS-based and of non-PaaS-based projects. Given our provisional focus on feedback, key questions in these analyses were how projects differed in the use of feedback and how these differences were related to PaaS. Memo writing [13, p. 72] was a key activity at this stage.

Focused coding was followed by theoretical coding [13]. At this stage, we considered affordance as a possible meta-theoretical lens because our data emphasized the *actions facilitated by PaaS*, rather than outcomes that would materialize irrespective of actions, such as in the following quote:

> *"You can implement PaaS and do everything exactly as you did before. Then you wouldn't see any change. But PaaS allows you to do things differently. It opens up doors that were closed before. And that is how it can impact your software development process a lot." (Developer-and-owner, BPMCo)*

Combining the GTM and the affordance lens is consistent with existing affordance research [68] and with recent suggestions that meta-theoretical lenses can effectively guide the later phases of grounded theorizing [8, p. 3, 13, p. 16, 64, p. xiii]. The affordance lens provided a device for organizing, refining, and expanding our themes in four important ways. First, given the focus on actions in the affordance lens, we looked for a "theoretical re-description" [12, p. 8] of the actions in our data. Themes such as feedback, learning, and agility led us to the theoretical re-description of software development as a collective learning effort. Second, the affordance literature urged us to aggregate the numerous affordances of low granularity to a limited number of more abstract categories [12, 53]. We followed Strong et al. [68, p. 74] in aggregating affordances that were "functionally the same", i.e., that led to the same outcomes. For instance, we aggregated our initial codes "trying out things" and "triggering immediate customer feedback" (see Table 3) to the category "triggering continuous feedback" because both codes refer to immediate

and frequent feedback as an outcome. Third, the affordance lens invited us to explore the conditions related to each affordance [12, 68]. We coded which affordances were actualized in which team. Quite surprisingly, our data suggested that all PaaS-based teams had actualized nearly all affordances at least to some extent after relatively short time. We therefore delved into instances where teams had only partially [12, 68] actualized a particular affordance. We explored the conditions that could explain partial actualization by comparing potential relationships, such as affordance dependencies [68, p. 74], against our coding of affordance actualizations per team and against the longitudinal evidence depicted in our visual maps. The result of these activities was substantive theory [34] that explains how, why, and when particular affordances arise for software development teams from their use of PaaS.

RESULTS

Figure 2 visualizes our emerging theory. We found two capabilities of PaaS to be particularly salient in our data: *rapid elasticity* and *abstraction*. Four affordances arose in the relation of these capabilities and software development teams: *shaping environments, reusing software services, self-organizing, and triggering continuous feedback*. While the first two were basic affordances, self-organizing and triggering continuous feedback were higher-level affordances that depended on the immediate concrete outcomes from actualizing basic affordances. The degree to which teams actualized higher-level affordances depended on the work environment characteristic of *self-contained tasks*. Although the actualization of the reusing software services affordance made tasks more self-contained, functional sub-teams and architectural dependencies sometimes inhibited self-contained tasks and, hence, the actualization of higher-level affordances. The key impact from actualizing the affordances was enhanced *agility*. We proceed by presenting the categories related to capabilities, affordances (including outcomes and impact), and work environment characteristics.

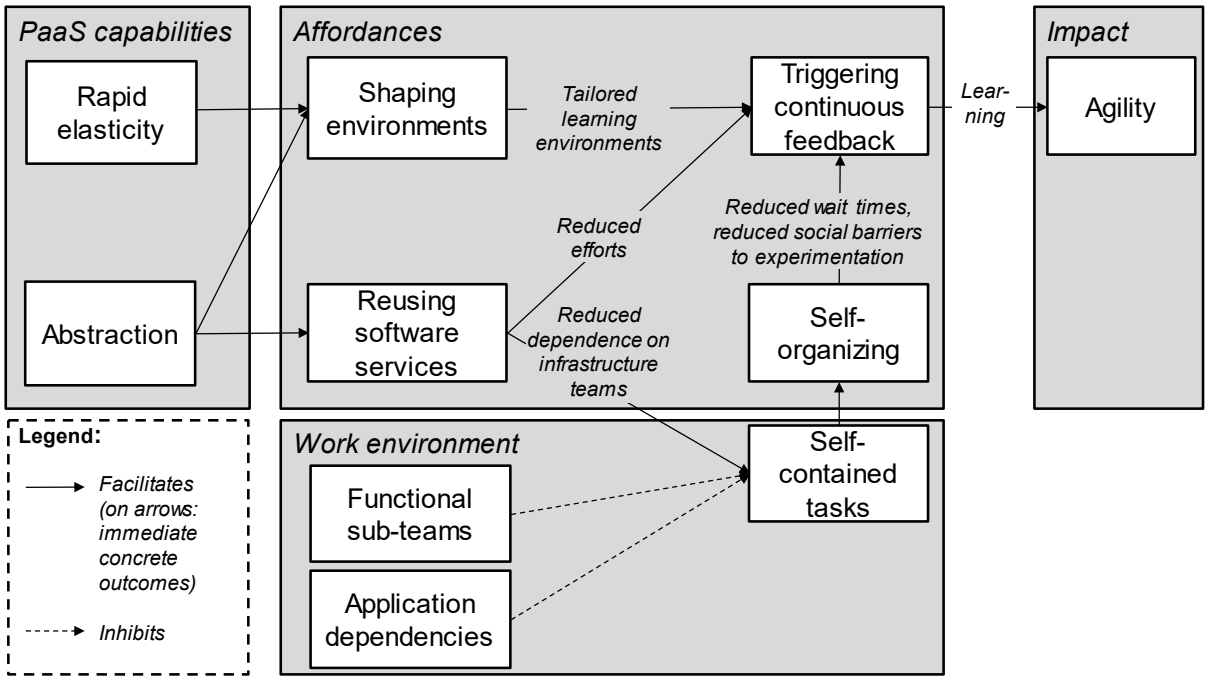

Figure 2: Emerging Theory

Technological Capabilities

Although PaaS offers a range of technological capabilities, two emerged as particularly important in enabling the affordances presented in this paper: rapid elasticity and abstraction. Table 3 provides definitions and example quotes. *Rapid elasticity*, an essential characteristic of any cloud technology [56], is the capability of immediately changing the scale of the underlying virtualized hardware, such as processors and memory. *Abstraction* is the capability of suppressing the details that are encapsulated in the cloud service [71, p. 187]. PaaS abstracts away the inner workings of application environments, including hardware virtualization, application servers, database servers, patches, service and code integration, load balancing, security and redundancy, deployment, roll-back after failed deployment, and configurable software components. Abstraction implies that developers are relieved from care of such issues below the application layer, as one informant put it: *"I need to care almost only about my application layer" (DevCo, manager)*. Abstraction follows from the essential cloud computing characteristic of on-demand self-service [56]. Since PaaS provides application environments as an automatic, unilateral service, the

inner workings of this provisioning process and of its technical components are hidden to the consumers of the service (i.e., software developers).

Table 4: PaaS Capabilities

Category	*Definition*	*Example quotes*
Rapid elasticity	The capability of immediately changing the scale of the underlying hardware	"Scaling is a matter of pushing a button." "As a user I can say: I'd like to have five instances. Now I need more RAM, more CPU. Or less." "You can scale extremely fast."
Abstraction	The capability of suppressing the details that are encapsulated in the cloud service	"You never have to update scripts or applications. They are always up to date." "When we start a project, we never talk about servers, hardware, Windows version, Linux, how many databases, how much memory and so on. We don't set up development servers or infrastructure. We never talk about IT."

Affordances and Immediate Concrete Outcomes

Four affordances arose in the relation of technological capabilities and software development teams. Table 5 shows the four affordances along with conditions for their actualization, the actions needed to actualize the affordance, and immediate concrete outcomes, illustrated by example quotes. We next introduce each of the affordances.

Shaping environments

The combination of the rapid-elasticity and abstraction capabilities allowed teams to *shape environments*, i.e., to easily align application environments to their needs. Specifically, PaaS allowed teams to easily change the machine power, the software configuration (e.g., choice of database servers), and the data of their environments. The ability to shape environments was useful in *learning environments*, i.e., in environments that teams used for the purpose of generating feedback. Examples of learning environments include development, test, and integration environments. An immediate concrete outcome from shaping environments were *tailored learning environments*, i.e., environments that were tailored to the learning needs of the team (see also the arrows in Figure 2 and the last column of Table 5).

Table 5: Affordances that PaaS offers to Software Development Teams

Definition	*Conditions*	*Actions needed to actualize the affordance*	*Immediate concrete outcomes with example quotes*
Shaping environments			
Aligning application environments with one's needs.	*Technology:* Abstraction, rapid elasticity; *Context*: Financial resources for environments available	Teams create representative or deliberately modified environments.	*Tailored learning environments*: "[With PaaS], I can immediately go into an environment … that is absolutely identical to the production environment." "Let's do a load test only with one instance and 1 GB of RAM. … With PaaS, this is just one command line. Now let's test it with hundreds of instances with 6 GB each."
Reusing software services			
Making use of existing software services, typically services developed by the PaaS provider.	*Technology*: Abstraction; *Actors*: Individuals are familiar with PaaS-based development principles	Teams use the PaaS services (e.g., setting up application environments), instead of developing the services (e.g., developing scripts for setting up environments) or doing the work manually (e.g., purchasing machines).	*(1) Reduced efforts*: "It is very fast and cheap to move a field." "[With IaaS] we have an effort of half a day to configure a server. With Heroku, it's two clicks." *(2) Reduced dependence on infrastructure teams:* "If you want this or that service …, you don't have to run each time to a system administrator and ask." "You don't need another person to take care of infrastructure. The developer himself can arrange for the infrastructure … He does not have to involve other people to deal with the issues below the application layer …Hence, the developer is really independent."
Self-organizing			
Organizing work through local actions and interactions.	*Outcomes from lower-level affordance:* Reduced dependency on infrastructure teams; *Context*: Self-contained tasks	Teams make decisions spontaneously and autonomously, instead of following external order.	*(1) Reduced wait times:* "If we have to set up a server [without PaaS], we need to order a server. This may easily take a few weeks … This falls away [with PaaS]." *(2) Reduced social barriers to experimentation*: "When we were not yet in the cloud, I sometimes thought that a Redis server [a particular database technology] would be great. Then we talked about it [with administrators]: 'Yeah, but setting up a Redis server (irritated). Would you need that for other projects as well?' Now in the cloud, I can just try it."
Triggering continuous feedback			
Provoking immediate feedback as early and as often as needed.	*Outcomes from lower-level affordances*: tailored learning environments, reduced efforts, reduced social barriers to experimentation, reduced wait times	Teams trigger feedback on demand, such as by deploying immediately, frequently showing version of the software, and deliberately experimenting with technologies.	*Learning*: "You deploy to integration or test environments on a daily basis. As a consequence, you [immediately] note many issues that would otherwise fall on your feet two weeks later if you deployed only every two weeks. This definitively increases productivity." "Each [feedback] loop puts in a better position… By having access to the software from the very beginning, by trying it out and doing their work with it, customers change their requirements fundamentally over time."

There were two types of tailored learning environments. The first were *representative environments*, i.e., environments that were as similar as possible to production environments. Our informants reported that in non-PaaS settings, testing was often hampered by differences between testing and production environments in terms of machine power, configuration (e.g., database products), and data (e.g., simplified test data differing from production data), such as in the non-PaaS-based TelcoNews team:

> *"After testing on integration, we could assume that it will also work on production. However, this isn't always true. There are cases in which the environments still differ, even if only slightly …, like a cache being configured differently." (TelcoNews, developer 1)*

PaaS facilitated creating representative environments because rapid elasticity reduced constraints on machine power while abstraction helped to make configuration and data identical:

> *"[Without PaaS] it is difficult to have environments that are identical to your production environment. With PaaS, the environments are exactly identical."(PaaSCo, administrator)*

> *"Salesforce offers you a sandbox called Full Sandbox. That's a 1-to-1 copy of your production environment. All the data is there." (ForceCo, developer 1)*

A second type of tailored learning environments were *deliberately manipulated learning environments*. These environments were intentionally different to enable testing under alternative conditions:

> *"Testing can be also much easier with PaaS. For example, you can do scalability tests in a dimension like never before. You can start hundreds of servers at once … and test your application on a very high scale. To do something like this on-premise, you would have to buy hundreds of servers, which would, of course, not be affordable. With PaaS you can start those virtual servers for a short period of time and stop them." (BPMCo, owner-developer)*

With the exception of TelcoAPI, who did not use any learning environments, all PaaS-based teams actualized the shaping environments affordance in at least some projects. Yet financial constraints prevented some teams from actualizing the affordance in some projects. For instance, informants from ForceCo and CRMCo reported that in some projects, the customers were not willing to bear the fees that the PaaS provider charged for representative test environments. This illustrates that actualizing this affordance involved a choice, indicating that shaping environments was indeed a *potential* for action.

Reusing software services

The abstraction capability provided a second affordance: *reusing software services*. These software services comprised process services (i.e., software services that support the process of software development) and product services (i.e., software services that become part of the developed software product). Process services included services for setting up virtualized infrastructure, services for configuring and updating application and database servers, and services for deploying code and rolling back. Product services included PDF creators, user interface components, objects, fields, and workflows.

As the term *reuse* suggest, PaaS enabled teams to make use of existing software services, typically services developed by the PaaS provider. This yielded two immediate concrete outcomes: reduced efforts and reduced dependence on infrastructure teams. Reusing process or product services *reduced efforts* because it eliminated the work that was abstracted away by the PaaS service. For instance, reusing software services eliminated efforts for deployment automation, configuration, and requirements analysis:

> *"As soon as you are finished, you can deploy the code with one single command. Of course, this is also possible with IaaS or an on-premise solution, but you would first have to set up a platform that does all this." (BPMCo, owner-developer)*
>
> *"With on-premises you have to think about network configuration, load balancing, redundancy, and so on. … With PaaS these things are … automated." (PaaSCo, manager)*
>
> *"[In non-PaaS software development] one used to lose a lot of time with basic things, such as usability, user authentication, and infrastructure. That's the major change with Salesforce [during requirements analysis]. You can talk 90% of the time about business processes. This makes the phase much shorter." (ForceCo, manager 1)*

Nearly effortless deployments with PaaS contrasted with, for example, the non-PaaS-based team TelcoNews, where each deployment took *"2-3 hours" (TelcoNews, developer 1)*.

By reusing process services, software development teams *reduced their dependence on infrastructure teams* because the automatic process services substituted for the manual work that was traditionally

performed by infrastructure teams. Thus, software development teams no longer depended on infrastructure controlled by these teams and, hence, obtained easier access to resources such as servers:

> *"I also work in a [non-PaaS-based] project. There I realized how difficult it is to get a server, even a virtual server… With PaaS this is a lot easier." (SecureCo, developer 1)*

In a similar vein, an informant from Telco cynically narrated how, in a current non-PaaS-based project, one of his colleagues *"sends Excel sheets with port activation requests to an anonymous email address just to wait until some guy activates the port one month later and you can do the deployment" (TelcoAPI, manager).* In contrast, by reusing the process services provided by PaaS, developers were able to arrange for the required infrastructure themselves:

> *"[The developer] can … set up this environment. He can rapidly add new services. He can rapidly create a test environment. He can do all this on his own". (TelcoMedia, manager)*

Although, in principle, developers could also set up their infrastructure without PaaS, such as by setting up physical machines themselves or by using IaaS, many still depended on the help by infrastructure teams because *"developers often do not have the knowledge for setting up environments and deploying code"* (DevCo, manager). Since PaaS abstracted away issues related to application environments, developers did not require such knowledge anymore in order to set up environments themselves.

Reduced dependence on infrastructure teams gave rise to an important change in the work environment: the tasks of many teams became *self-contained* [30] (see the arrow from *reusing software services* to *self-contained tasks* in Figure 2). That is, many teams now had control over the use of all resources (infrastructure, knowledge, and software) required to independently complete their task.

Much like shaping environments, reusing software services was a *potential* for action. Teams were able to choose whether to reuse the process and product services provided by the PaaS product. Although there was a choice, all 14 PaaS-based teams in our sample actualized the reusing software services af-

fordance to a large extent. However, our informants reported that they had to learn PaaS-based development principles before they could "appropriately" [12, p. 7] actualize the affordance (see the third column in Table 5). Specifically, many informants mentioned a learning process of a few weeks during which they learned principles such as stateless development, no use of file systems, and microservice-based architecture. Although this learning process was necessary, our informants generally found this process to be manageable, or *"not a big deal"* (TelcoOrder, developer 1).

Self-organizing

Self-contained tasks enabled by the reusing software services affordance gave rise to a third affordance: *self-organizing*. Teams were able to make local, spontaneous decisions such as when to deploy, how to coordinate, and what technologies to use. They were able to make these decisions themselves because they did not depend anymore on other teams that could impose order. Accounts from a variety of teams illustrate how self-contained tasks, resulting from the reduced dependency on infrastructure teams, facilitated self-organization. Members of the TelcoAPI team narrated how infrastructure teams imposed processes on them and how reusing software services with PaaS liberated them from these processes:

> *"Here at Telco, setting up traditional infrastructure means speaking with a DevOps guy to get an application server. Then I have to ask the firewall guys to open a port... Then I have to consider firewall rules and security issues. So, in order to publish a service with on-premises infrastructure, I need to do four things. With PaaS I'm done in 10 minutes." (TelcoAPI, developer 2)*

> *"[With PaaS], you have your personalized environment, including services, ready within 3 minutes, entirely sidestepping your organization." (TelcoAPI, manager)*

A developer from the TelcoMedia team narrated how self-contained tasks allowed ad-hoc decision making about deployments:

> *"A lot changed [in our team structures]… We now don't have anyone who is responsible for the hardware, the server, and the databases. We just need someone who manages the PaaS, which is really not a lot of work. This can be done either by a developer or by the project manager. We also do not need any integrators because every developer can deploy directly to the PaaS. … The structures in our team evolved ad hoc, how it made sense to us." (TelcoMedia, developer 1)*

Reduced dependencies on other teams also gave developers greater freedom to choose the technologies that they found useful, rather than technologies that others had approved:

> *"You don't have to take what your firm prescribes. You can say I take this service from here and this service from there. I do this on my own. People can work with what they love and what they need." (TelcoAPI, manager)*

Self-organizing yielded two important immediate concrete outcomes: *reduced wait times* and *reduced social barriers to experimentation*. By organizing work inside the team, PaaS-based teams did not have to wait for the actions of others, or *"external blockers"* (PaaSCo, manager). In contrast, non-PaaS-based teams often waited a long time for others' actions, such as in the non-PaaS-based TelcoNews team:

> *"We completed the work on the software six months ago. Now we're still waiting for the infrastructure to be ready." (TelcoNews, developer 2)*

Self-organizing also lowered social barriers to experimentation because developers did not have to fear that their requests for work from other teams would provoke negative reactions:

> *"If you use an on-premises environment only for a quick experiment and then throw it away, the administrator, who spent several hours setting it up, will murder you. With PaaS you simply create such a throwaway environment with a single click." (PaaSCo, manager)*

Although the outcomes from reusing software services facilitated self-organizing, teams could choose to what extent they wanted to actualize this affordance. For instance, it was, in principle, possible to reuse the PaaS services but to stick to externally imposed roles and processes, such as by asking infrastructure teams to set up PaaS environments or by following plan-based software development approaches. As one informant put it: *"You can plan yourself to death for half a year and then install everything on PaaS"* (TelcoOrder, developer). Hence, although reusing software services facilitated self-organization, teams were able to choose whether and to what extent to actualize this affordance, showing that self-organizing was a potential for action.

Triggering continuous feedback

The key affordance that arose in the relation of PaaS and software development teams was *triggering continuous feedback.* That is, the PaaS-based teams in our sample were able to provoke immediate feedback as early and as often as needed. Feedback included internal feedback and external feedback (see section 2.2 for definitions). Instances of triggering continuous internal feedback included: developers experimenting with alternative technologies when they wished to do so; developers perceiving the results of deploying code to representative test environments immediately after writing the code; customers immediately seeing a version of a software based on their requirements; customers immediately experiencing the consequences from their changes to the software configuration. Instances of triggering continuous external feedback included: developers showing new features to customers immediately after developing them to obtain feedback; developers deploying new features to production environments immediately after developing them to obtain feedback from end users.

The potential to trigger continuous feedback arose because the outcomes from actualizing the other three affordances helped tear down temporal, technical, and social barriers to feedback. *Reduced effort and reduced wait times* helped tear down temporal barriers to feedback. For instance, with reduced development and administration efforts and with eliminated wait times for deployment, the time from articulating a requirement until seeing the software in action was often substantially shortened:

> *"An enormous difference to classical software development is that you can set up a working prototype in no time… Thus, you get feedback from the beginning... You continuously show the application and obtain feedback. Because you can and because it is very easy and quick." (ForceCo, manager 1)*

> *"When you want to hand over code for testing, … you can quickly create a test environment on Heroku, deploy to it, and look at it. As a developer, you are able to much faster show something to customers. You have a shorter feedback loop." (DevCo, manager)*

Tailored learning environments helped reduce technical barriers to feedback by providing teams with the technical environments that fit their learning needs.

"With PaaS, each developer could deploy his code on his own test environment, which is an exact copy of the production environment. If the code works, it will also work on the production environment." (ShareCo, owner-developer)

Reduced social barriers to experimentation helped reduce social barriers to feedback:

"PaaS allows developers to try out different services quickly, play around with them, and decide for the best. Try this in with on-premises—impossible. … System administrators would go crazy if you asked them continuously to install and uninstall certain services for testing." (PaaSCo, manager)

The key outcome from triggering continuous feedback was *learning*. Continuous feedback helped customers to refine their ideas about their requirements:

"Sometimes, the customers don't know what they want. Then the feedback [from working with a prototype] often results in different or new requirements." (ForceCo, manager 2)

"With PaaS, we can create many small prototypes and test instances that we show the customer on demand… This greatly improves … the customer's understanding." (DevCo, manager)

Continuous feedback also helped developers to learn from mistakes made during coding:

"If a developer deploys himself and then notes that something does not work because of his code, then he realizes this himself and learns. PaaS allows us to think more and more along these lines." (DevCo, manager)

The ability to continuously promote learning through continuous feedback was important for all teams in our sample because they shared the goal of *agility* [19]. Continuous feedback enabled them to continuously create change (e.g., by quickly implementing a new requirement, by quickly trying out technologies, by deploying immediately) and to immediately learn from it (e.g., by seeing how useful the new requirement, the technology, or the code change was).

As with the other affordances, actualizing the triggering continuous feedback affordance involved a choice. We next elaborate on the conditions under which teams actualized higher-level affordances.

Work Environment Characteristics

While all PaaS-based teams actualized the basic affordances shaping environments and reusing software services to a large extent, we found more variation between teams, and within teams over time, in

the degree to which teams actualized the high-level affordances self-organizing and triggering continuous feedback. Teams actualized the self-organizing affordance, and hence the triggering continuous feedback affordance, to a lesser degree when two context factors inhibited self-contained tasks: *application dependencies* and *functional sub-teams* (see the dashed arrows in Figure 2 and the definition and example quotes provided in Table 6).

Table 6: Work Environment Characteristics

Category	*Definition*	*Example quotes*
Self-contained tasks	A state in which a team has control over the use of all resources (infrastructure, knowledge, and software) that are required to independently complete its task	"You get teams that can operate highly autonomously." "We deploy to production because the operations team has been uncoupled." "You have a very broad knowledge. You could do an entire project on your own."
Application dependencies	The extent to which an application has interfaces to other applications	"PaaS is great if you don't have any mainframes in your basement with which the software needs to communicate." "We learned that we have to be realistic. Recently, we had the case of a software that strongly depends on data from certain back-end systems. In cases like this, we cannot operate in the same way as in our typical PaaS projects."
Functional sub-teams (vs. cross-functional teams)	A team design where teams are subdivided into specialized sub-teams	"The Mango team is responsible for the mobile application and the web interface. … But sometimes the web interface was not compatible with the back-end, which is managed by the Kiwi team. So in this cycle, the Mango team started deploying in cooperation with the Kiwi team. But then the Durian team, who is responsible for the desktop clients was not ready. So they could not deploy."

As laid out in the section on the reusing software services affordance, the tasks of many teams became self-contained by reusing PaaS services because the teams eliminated their dependence on the infrastructure and knowledge possessed by infrastructure teams. Self-contained tasks, in turn, enabled self-organization. For tasks to be self-contained, teams need to have control over all resources required for their tasks [30]. These resources include not only infrastructure and knowledge but also applications. Yet some tasks were characterized by strong *application dependencies*. In such tasks, the interfaced application was an important resource outside the team. When the interfaced application was under control of a team that followed external order (e.g., by following prescribed waterfall methods), then the focal team also had to adhere to external order, such as in the case of the TelcoNetwork team:

> *"We move outside the [regular three-months] release cycle [formally prescribed by Telco]… unless we develop enhancements for projects in which Core systems [i.e. systems which are developed based on the waterfall model] are affected. In that case, we rely on data from Core systems and we deploy within the official release cycle. But for other features, we move outside [the waterfall development process] and we release how we want." (TelcoNetwork, developer 1)*

Second, self-contained tasks were also inhibited by *functional sub-teams*. For instance, after TelcoMedia had fully actualized self-organization and continuous feedback for some time, managers of the TelcoMedia team decided to divide the team into four sub-teams with functional specialization, such as a front-end team and a back-end team. As a consequence, the tasks of these sub-teams were not self-contained anymore given that many tasks affected both front-end and back-end and, hence, involved people outside a particular sub-team. With such increased dependencies, the team had to abandon the highly self-organized way of working of the first project phase and adopted a more central coordination approach with rigid roles and fixed two-weeks cycles. With this time-boxed approach imposed from outside the sub-teams, the sub-teams did not actualize the potential to trigger continuous feedback anymore in a number of areas. An architect lamented that this team design slowed down the project:

> *"We didn't really want this. But the external vendor wanted it. [They] developed the back-end and they did not want to give up control over the back-end. That was the reason [for this team structure]… To be sincere, this really blocks us and it is inefficient." (TelcoMedia, architect)*

In contrast, BPMCo was also a large team of over 50 people but remarkably retained its ability to continuously trigger feedback. They strongly relied on self-contained tasks assigned to *"relatively small, autonomous [sub-]teams"*, *"typically [of] a back-end and a front-end developer"* (BPMCo, owner-and-developer). BPMCo fully actualized the self-organization and continuous feedback affordance and *"deploy[ed] each feature … as soon as it [was] ready, totally dynamically, sometimes multiple times a day"* (BPMCo, owner-and-developer).

Alternative Explanations or Predictions

During our data analysis, we examined several alternative explanations and predictions for affordances or, more broadly, the impact of PaaS on software development. They emerged from our literature review

or from our data analysis but were discarded as our theoretical sampling and analysis unfolded. For instance, although some informants conjectured that PaaS affords self-organizing and triggering continuous feedback only if the dominant organizational software development method is an agile method, we also found these affordances to be actualized in organizations that prescribed waterfall development. Table 7 provides a brief discussion of this and other explanations or predictions.

Table 7: Alternative Explanations or Predictions

Category	*Conjecture*	*Evidence*
Intentions behind PaaS use	Teams actualize higher-level affordances only if greater agility is their intention behind PaaS use.	Teams actualized higher-level affordances irrespective or their initial intentions behind PaaS use. Teams, such as TelcoMedia and TelcoNetwork, that had not anticipated the action potential for self-organizing and continuous feedback before using PaaS, quickly perceived and actualized these action potentials after they started using PaaS.
External vs. internal provider	The impact of PaaS depends on whether the PaaS provider is external or an in-house unit.	Teams actualized higher-level affordances irrespective of whether PaaS was provided by an in-house unit (like in the four PaaS-based teams at Telco) or by an external firm (like in all other teams studied).
Team size	Only small teams are able to actualize higher-level affordances.	Teams actualized higher-level affordances irrespective of team size. Even large teams, such as BPMCo with over 50 members and TelcoMedia with over 30 members, actualized these affordances as long as the sub-teams were cross-functional.
Project duration	PaaS use correlates with agility because PaaS is primarily chosen for short projects.	PaaS-based teams actualized higher-level affordances and were highly agile even in long, large projects. For instance, TelcoMedia was a multi-year project characterized by high agility, as evidenced, for instance, by lead times for new features of two weeks. Such lead times were rare in other teams at Telco.
Organization size	Only teams in small organizations actualize higher-level affordances.	Teams actualized higher-level affordances irrespective of the size of organizations. Since the use of PaaS helped teams to decouple themselves from their organizations, even formal cultures typical for large organizations, such as Telco and SecureCo, hardly inhibited higher-level affordances.
Dominant organizational software development method (DOSDM)	Teams actualize higher-level affordances only if the DOSDM is an agile method.	Teams actualized higher-level affordances irrespective of the DOSDM. For instance, although Telco and SecureCo had strong traditions in waterfall development, teams quickly started self-organizing and triggering continuous feedback after adopting PaaS. Since the use of PaaS helped teams to decouple themselves from their organizations, strong traditions in waterfall development, like at Telco and SecureCo, did not inhibit higher-level affordances.
Network effects	The impact of PaaS depends on network effects. It is higher when a PaaS product attracts many customers and independent software vendors.	In our data, the impact of PaaS on software development practices hardly depended on network effects. The affordances arose primarily in a dyadic relationship of a PaaS provider and a software development team. Although the teams sometimes used services (e.g., PDF creators) offered by independent software vendors, it was in particular the use of services offered by the PaaS provider (e.g., deployment features, objects, fields, workflows) that had a transformative impact on software development practices in the teams that we studied.

DISCUSSION

In this research, we set out to study the impact of PaaS on software development. The underlying motivation was the lack of empirical studies that examine how software development practices change with the use of PaaS. The key outcome from our study is an emerging theory, grounded in evidence from 16 teams, that describes the affordances offered by PaaS and that explains how, why, and when these affordances arise. The key affordance is triggering continuous feedback, and its key impact is enhanced agility. Implications from our theory are (1) that the use of cloud technology can help accelerate technology-mediated collective learning processes under certain conditions, (2) that PaaS is an enabling technology in the transition to practices advocated by many current software development movements, and (3) that the actualization of affordances can be inhibited by work environment characteristics that conflict with the logic underlying the affordances. We next discuss these and other implications for the literatures on cloud computing, software development, and affordances in more detail.

Implications for the Cloud Computing Literature

The Impact of PaaS on Software Development: Beyond Efficiency Gains and the Outsourcing Metaphor

Our findings related to basic affordances echo claims in the cloud literature about efficiency gains enabled by PaaS. For instance, our findings related to the shaping environments affordance resonate with Lawton's assertion that "[b]uilding, debugging, testing, and deploying applications in the same environment … reduces project risk by eliminating problems caused when programs are developed in one setting but run in another" [49, p. 14]. Our findings related to the reusing software services affordance are broadly in line with the view of PaaS as "a way for companies … to help contain or drastically reduce IT expenditures" [26, p. 22]. However, our findings go beyond the prevailing focus on particular efficiency gains, or the idea that PaaS "may affect aspects of system testing and implementation phases, but have little impact on the design and development phases" [81, p. 48].

Our study adds that actualizing basic affordances not only yields efficiency gains during particular phases; it also enables the transformation of work practices towards greater agility that is delineated in our two higher-level affordances. The first of these higher-level affordances is *self-organizing*. By reusing the software services provided through an on-demand self-service, software development teams reduce their dependence on infrastructure teams, which makes the work of software development teams self-contained and thereby facilitates self-organizing. Since these teams do not depend anymore on the knowledge and infrastructure possessed by infrastructure teams, they can more easily make local, spontaneous decisions such as when to deploy, how to coordinate, and what technologies to use. The literature on PaaS has rarely explicitly described the potential of PaaS to facilitate self-organizing, although the idea that PaaS affords self-organizing is somewhat implicit in the assertion that mPaaS might enable end user computing [81]. Yet end user computing (i.e., business users changing applications themselves) was only one, and not the most salient, facet of self-organizing facilitated by PaaS that surfaced in our data. Although users sometimes performed minor changes to applications themselves, the much more salient pattern of self-organizing behavior was that software development teams leveraged PaaS to make autonomous decisions about issues such as deployment, team coordination, and technologies.

It is insightful to relate our findings on self-organizing to the broader cloud computing literature. Some work on SaaS alludes to the potential for cloud users to self-organize their use of computing resources but gives it a rather negative connotation through terms such as stealth adoption or shadow IT [35, 82]. This contrasts with our informants' positive perspective, one of whom viewed the potential to self-organize the software services underlying his work as *"awesome democratization"* (TelcoAPI, manager). The idea that cloud technology enables self-organizing contrasts also with the *outsourcing metaphor* that is sometimes used in cloud computing research. While the notion of outsourcing sensitizes for heightened boundaries and arduous coordination with external parties [23, 43], the PaaS-based teams in our sample achieved quite the opposite: They reduced their dependence on and their need to coordinate with other

units because the work of these units became an on-demand self-service. Paradoxically, by relying on an on-demand self-service, software development teams abdicated control over the provisioning of infrastructure but gained control over the choice and use of infrastructure, allowing developers to decide what technologies to use and when to make what changes. This potential for self-organizing existed irrespective of whether the PaaS provider was internal or external (i.e., outsourcing). These findings echo voices [65] that have warned against literally transferring ideas from outsourcing research to cloud computing.

A second higher-level affordance emerging from our analysis is *triggering continuous feedback*. The outcomes from actualizing the other three affordances helped teams to tear down the temporal, social, and technical barriers to feedback, allowing teams to trigger nearly immediate feedback when needed and to learn from it. With few exceptions [71], academic studies on cloud computing have rarely recognized its potential to accelerate technology-mediated feedback processes and to thereby enhance agility.

Closest to this idea is perhaps Venters' and Whitley's assertion that "the scalability of cloud services allows the trialling of niche services in an agile manner with low risk" [71, p. 190]. Our emerging theory expands this idea in three ways. First, it is not only the scalability (or rapid-elasticity) capability but also the abstraction capability that helps accelerate feedback. Abstraction helps reduce efforts and liberate from dependencies, which reduces temporal and social barriers to feedback. Second, these capabilities enable quicker feedback and, hence, agility by allowing not only to trial different services but also to continuously make changes to a particular service and to observe the outcomes (such as by changing configuration or source code and immediately testing the software in production-like conditions). Third, our emerging theory recognizes that teams may not be able to realize the potential to enhance agility under all circumstances. Two work environment characteristics, application dependencies and functional subteams, can inhibit self-organizing and, hence, triggering continuous feedback. Thus, whether and how teams can harness the potential for greater agility offered by PaaS is likely to depend both on exogenous factors (i.e., application dependencies) and on team design (i.e., choice of cross-functional teams).

Although we developed a substantive [34] PaaS-specific theory, we believe that our theory can serve as a first step towards formal theory [34] of the transformative impact of cloud technology on technology-mediated collective learning processes. Two empirical settings may help illustrate the application of the theory: IaaS-based software development and SaaS implementations. Relative to PaaS, IaaS offers a similar level of rapid elasticity but a lower level of abstraction. Applying our theory to *IaaS-based software development* would thus suggest that with lower abstraction, IaaS provides qualitatively similar but somewhat weaker affordances (i.e., actions are not facilitated to the same extent as in the case of PaaS). For instance, the self-organizing affordance is weaker because IaaS eliminates the need for the physical infrastructure possessed by infrastructure teams but not for their knowledge about the provisioning of the infrastructure. The triggering continuous feedback affordance is also weaker given the social and temporal barriers to feedback that remain due to the weaker self-organizing affordance and due to the higher efforts for managing infrastructure and building application components.

Our theory may also be applied to *SaaS implementations* given that, like software development, software implementations are technology-mediated collective learning processes (e.g. teams learning new ways of configuring the software and of using it in their work routines [50, 59]). Since SaaS offers a higher level of abstraction than PaaS, it may provide stronger affordances for self-organizing and triggering continuous feedback than PaaS, in particular to business users. Whereas the business users in the PaaS-based teams in our sample rarely configured the software themselves, SaaS may enable business users to make spontaneous, local, and autonomous decisions about the choice, configuration, and use of software without adhering to external order. They may also trigger continuous (internal) feedback by trialling a variety of SaaS services [71, p. 190] and by immediately experiencing how they could perform their work with the help of these services. Extrapolating our finding on the inhibiting role of work environment characteristics, we expect that the affordances for self-organizing and continuous feedback are

less pronounced in the case of enterprise software, where dependencies on other teams are relatively strong, than in the case of team-level software, such as collaboration software. Although these ideas have not been validated through our study, they are intended to illustrate how researchers can build on the key idea behind our emerging theory: the idea that the abstraction and rapid-elasticity capabilities of cloud computing technology help remove the barriers to feedback that otherwise slow down technology-mediated collective learning processes.

Implications for the Software Development Literature

Our study offers insights into the transition process towards self-organizing and continuous feedback practices, which is a key focus of the software development literature and of current software development movements. The transition to *self-organizing* practices is a key concern of the DevOps movement, which aims to break down social barriers to collaboration by promoting collaborative norms in united teams of developers and administrators [42, 63, 77]. While this often presents a relatively slow process of learning and cultural change, the PaaS-based teams in our sample followed a different, relatively quick process with, however, a similar outcome. Rather than to promote collaboration, they chose to eliminate administrators by substituting their work with an automatic on-demand service. This quickly eliminated the frictions between developers and administrators and enabled teams to make spontaneous, local decisions about the use of infrastructure. Thus, our study provides the insight that PaaS can facilitate the transition to self-organizing practices because its abstraction capability allows developers to reuse software services, which reduces the dependence on and the order imposed by others.

The transition to *continuous feedback* is also a key concern behind many current software development movements, such as the agile movement [39, p. 121], and the more recent CI/CD and lean software development movements [28, 41, 42, 62]. Research on these approaches has described the transition to continuous feedback practices as a tedious process that is fraught with technical challenges in particular

at the outset of projects and that demands significant top management support [17, 28, 48, 67]. In contrast, the PaaS-based teams in our sample actualized the continuous feedback affordance relatively early and easily, without significant management support. The use of PaaS helped these teams to tear down the temporal (wait times, large tasks), social (need to collaborate with infrastructure teams), and technical (heterogeneous environments) barriers to feedback. For these teams, the transition to continuous feedback practices was a relatively easy, natural process. This comparison of our findings with the tedious processes reported in the literature underscores a key assertion of our paper: that PaaS provides an affordance for continuous feedback. The use of PaaS can make it easier for teams to enact continuous feedback practices, although teams can also enact such practices without PaaS, albeit with greater effort. An important contribution of our paper is, hence, that we reveal the enabling potential of PaaS for software development practices that rely on continuous feedback.

Our findings on work environment characteristics resonate with insights provided by the software development literature. Similar to our finding that application dependencies inhibit higher-level affordances, research on CI/CD has found that application dependencies are a major problem in CI/CD adoption [48]. Moreover, in line with our observation that functional teams inhibit higher-level affordances, the literature on CI/CD advocates the use of cross-functional teams to implement CI/CD [41]. These commonalities are not surprising given that the use of PaaS facilitates the practices advocated by the CI/CD movement.

Implications for the Affordance Literature

Although the primary contributions of our study relate to the cloud computing and software development literatures, we also offer one insight to the affordance literature. While affordance studies explain the actualization of affordances primarily with actor characteristics (e.g., goals, skills) [50, 68], our study revealed that work environment characteristics can be important inhibitors for the actualization of affordances. Our teams were unable to fully actualize the self-organizing and, hence, the continuous feedback affordances when they were constrained by work environment characteristics—functional sub-

teams and application dependencies—that were at odds with the logic of self-organizing. With the exception of Strong and colleagues, who briefly mention work environment characteristics [68, p. 72], few scholars have incorporated work environment characteristics into affordance-based theories. Our study indicates that conflicts between the logic of affordances (e.g. the principle of self-organizing systems) and work environment characteristics (e.g. strong dependencies on the environment) can help explain variation in the actualization of affordances.

Limitations

Our study is not without limitations. First, unlike in a case study, the focus in our study was on breadth rather than depth. Given the lack of knowledge about PaaS, breadth was useful for uncovering themes that are salient across many PaaS-based teams, rather than idiosyncratic to a particular setting. Future research could focus on depth, analyzing one or few cases of PaaS-based software development over time. Such a case study could provide richer insights into processes of change. Second, our study relied on interview data only. Future research on the impact of PaaS could make greater use of triangulation, such as by corroborating interview statements with observational analysis, with the analysis of code versioning systems, and with more structured instruments that build on our findings. Third, although we examined a number of potentially inhibiting context factors through theoretical sampling and although our sample included a relatively large number of teams, all PaaS-based teams in our sample were overall satisfied with the affordances that PaaS provided to them. Future research could look out for cases of failed PaaS implementations to further sharpen the conditions under which the use of PaaS results into positive impact on agility. Fourth, most of our teams used a relatively homogeneous class of technologies (mPaaS and dPaaS services). Future research could compare a greater variety of technologies that differ in the level of abstraction (e.g., IaaS, container technologies, non-cloud based model-driven frameworks) to further disentangle the roles played by particular capabilities.

Practical Implications

With the growing adoption of PaaS, organizations are wondering what the value proposition of PaaS is and how they can harness it. Our study suggests that the primary value proposition of PaaS is agility. By enabling teams to trigger continuous feedback, PaaS can help to speed up the learning processes from which useful, innovative software results. Organizations should therefore consider adopting PaaS in projects that strive for rapid, software-based innovation and that are not hampered by complex dependencies on other applications. The mere adoption of PaaS may, however, not necessarily produce greater agility. Organizations should encourage self-organization and continuous feedback practices in these teams. Moreover, organizations should rely on cross-functional (sub-)teams in such projects. Given the role of application dependencies, organizations may start their PaaS journeys in isolated applications and then expand their use of PaaS along the network of related applications.

CONCLUSION

Although vendor studies point to a transformative potential of PaaS for agile practices and although means for enabling agile practices are of key interest to IS research and practice, the enabling potential of PaaS, and more generally of cloud computing, for agile practices has been largely unexplored. Our study unveils how two kinds of agile practices—self-organizing and continuous feedback—are enabled by two technical capabilities that are intimately related to the essential characteristics of cloud technology: rapid elasticity and the high level of abstraction offered by an on-demand self-service. Rapid elasticity and abstraction help teams to tear down the temporal, technical, and social barriers to feedback, resulting in accelerated technology-mediated collective learning processes and thus enhanced agility. While our study focuses on PaaS, future research may explore these mechanisms for other types of cloud technologies. More broadly, our study shows how cloud computing research can move beyond the outsourcing metaphor and explore the transformative value of cloud technology by paying close attention to the essential characteristics of cloud technology and the way how they facilitate particular practices.

REFERENCES

1. Anselmi, J., Ardagna, D., and Passacantando, M. Generalized nash equilibria for saas/paas clouds. *European Journal of Operational Research*, 236, 1 (2014), 326-339.

2. Austel, P., Chen, H., Mikalsen, T., Rouvellou, I., Sharma, U., Silva-Lepe, I., and Subramanian, R. *Continuous delivery of composite solutions: A case for collaborative software defined paas environments.* Paper presented at the 2nd International Workshop on Software-Defined Ecosystems. Portland, Oregon. 2015.

3. Beimborn, D., Miletzki, T., and Wenzel, S. Platform as a service (paas). *Business & Information Systems Engineering*, 3, 6 (2011), 381-384.

4. Bellomo, S., Ernst, N., Nord, R., and Kazman, R. *Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail.* Paper presented at the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Atlanta, GA. 2014.

5. Benedict, M. Coming to (your) terms with platform-as-a-service (paas). Bedford, MA: Progress Software, 2013, pp. 1-11.

6. Benlian, A., Hess, T., and Buxmann, P. Drivers of saas-adoption–an empirical study of different application types. *Business & Information Systems Engineering*, 1, 5 (2009), 357-369.

7. Benlian, A., Koufaris, M., and Hess, T. Service quality in software-as-a-service: Developing the saas-qual measure and examining its role in usage continuance. *Journal of Management Information Systems*, 28, 3 (2011), 85-126.

8. Birks, D.F., Fernandez, W., Levina, N., and Nasirin, S. Grounded theory method in information systems research: Its nature, diversity and opportunities. *European Journal of Information Systems*, 22, 1 (2013), 1-8.

9. Boh, W.F., Slaughter, S.A., and Espinosa, J.A. Learning from experience in software development: A multilevel analysis. *Management Science*, 53, 8 (2007), 1315-1331.

10. Brooks, F.P.J. *The mythical man-month: Essays on software engineering*. Reading, MA: Addison-Wesley, 1975.

11. Butler, D.L., and Winne, P.H. Feedback and self-regulated learning: A theoretical synthesis. *Review of educational research*, 65, 3 (1995), 245-281.

12. Bygstad, B., Munkvold, B.E., and Volkoff, O. Identifying generative mechanisms through affordances: A framework for critical realist data analysis. *Journal of Information Technology*, 31, 1 (2016), 83-96.

13. Charmaz, K. *Constructing grounded theory: A practical guide through qualitative analysis*. Thousand Oaks, CA: Sage, 2006.

14. Chen, L. *Towards architecting for continuous delivery.* Paper presented at the 12th Working IEEE/IFIP Conference on Software Architecture. Montréal, Canada. 2015.

15. Chen, L. Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software*, 128, June (2017), 72-86.

16. Chen, P.-y., and Wu, S.-y. The impact and implications of on-demand services on market structure. *Information Systems Research*, 24, 3 (2013), 750-767.

17. Claps, G.G., Svensson, R.B., and Aurum, A. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57, January (2015), 21-31.

18. CloudFoundryFoundation. Cloud foundry application runtime user survey. https://cloudfoundry.org/wp-content/uploads/2012/02/CFF_ApplicationRuntime_UserSurvey.pdf. Accessed Feb 27th, 2018.

19. Conboy, K. Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, 20, 3 (2009), 329-354.

20. Corbin, J., and Strauss, A.L. *Basics of qualitative research*. Thousand Oaks, CA: Sage, 2008.

21. Costache, S., Dib, D., Parlavantzas, N., and Morin, C. Resource management in cloud platform as a service systems: Analysis and opportunities. *Journal of Systems and Software*, 132, October (2017).

22. DaSilva, C.M., Trkman, P., Desouza, K., and Lindič, J. Disruptive technologies: A business model perspective on cloud computing. *Technology Analysis & Strategic Management*, 25, 10 (2013), 1161-1173.

23. Dibbern, J., Winkler, J., and Heinzl, A. Explaining variations in client extra costs between software projects offshored to india. *MIS Quarterly*, 32, 2 (2008), 333-366.

24. Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N.B. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85, 6 (2012), 1213-1221.

25. Espinosa, J.A., Slaughter, S.A., Kraut, R.E., and Herbsleb, J.D. Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, 24, 1 (2007), 135-169.

26. Fanning, K., and Centers, D.P. Platform as a service: Is it time to switch? *Journal of Corporate Accounting & Finance*, 23, 5 (2012), 21-25.

27. Faraj, S., and Sproull, L. Coordinating expertise in software development teams. *Management Science*, 46, 12 (2000), 1554-1568.

28. Fitzgerald, B., and Stol, K.-J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123, January (2017).

29. Fowler, M., and Highsmith, J. The agile manifesto. *Software Development*, 9, 8 (2001), 28-35.

30. Galbraith, J.R. *Organization design: An information processing view*. Reading, MA: Addison-Wesley, 1977.

31. Gartner. Size of the public cloud platform as a service (paas) market worldwide from 2015 to 2020. https://www.statista.com/study/31311/platform-as-a-service-statista-dossier/. Accessed Oct 6th, 2017.

32. Gass, O., Meth, H., and Maedche, A. Paas characteristics for productive software development: An evaluation framework. *Internet Computing, IEEE*, 18, 1 (2014), 56-64.

33. Giessmann, A., and Stanoevska, K. Platform as a service–a conjoint study on consumers' preferences. *The 33rd International Conference on Information Systems*, Orlando, FL, USA, 2012.

34. Glaser, B.G., and Strauss, A.L. *The discovery of grounded theory*. Chicago, IL: Aldine, 1967.

35. Haag, S., and Eckhardt, A. Shadow it. *Business & Information Systems Engineering*, 59, 6 (2017), 469-473.

36. Hahn, C., Huntgeburth, J., Winkler, T.J., and Zarnekow, R. *Business and it capabilities for cloud platform success.* Paper presented at the International Conference on Information Systems. Dublin, Ireland. 2016.

37. He, J., Butler, B.S., and King, W.R. Team cognition: Development and evolution in software project teams. *Journal of Management Information Systems*, 24, 2 (2007), 261-292.

38. Highsmith, J. *Adaptive software development: A collaborative approach to managing complex systems*. New York, NY: Dorset House, 2000.

39. Highsmith, J., and Cockburn, A. Agile software development: The business of innovation. *Computer*, 34, 9 (2001), 120-127.

40. Hilwa, A. Analyst watch: The evolving state of paas. http://sdtimes.com/analyst-watch-the-evolving-state-of-paas/. Accessed Feb 27th, 2018.

41. Humble, J., and Farley, D. *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Boston, MA: Pearson Education, 2010.

42. Humble, J., and Molesky, J. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24, 8 (2011), 6-12.

43. Jain, R.P., Simon, J.C., and Poston, R.S. Mitigating vendor silence in offshore outsourcing: An empirical investigation. *Journal of Management Information Systems*, 27, 4 (2011), 261-298.

44. Kauffman, S.A. *The origins of order: Self-organization and selection in evolution*. New York: Oxford University Press, 1993.

45. Krancher, O., and Dibbern, J. *Learning software-maintenance tasks in offshoring projects: A cognitive-load perspective.* Paper presented at the The 33rd International Conference on Information Systems. Orlando, FL. 2012.

46. Krancher, O., and Dibbern, J. *Knowledge in software-maintenance outsourcing projects: Beyond integration of business and technical knowledge.* Paper presented at the The 48th Hawaii International Conference on System Sciences. Kauai, HI. 2015.

47. Langley, A. Strategies for theorizing from process data. *The Academy of Management Review*, 24, 4 (1999), 691-710.

48. Laukkanen, E., Itkonen, J., and Lassenius, C. Problems, causes and solutions when adopting continuous delivery—a systematic literature review. *Information and Software Technology*, 82, February (2017), 55-79.

49. Lawton, G. Developing software online with platform-as-a-service technology. *Computer*, 41, 6 (2008), 13-15.

50. Lehrig, T., Krancher, O., and Dibbern, J. *How users perceive and actualize affordances: An exploratory study of collaboration platforms.* Paper presented at the The 38th International Conference on Information Systems. Seoul, South Korea. 2017.

51. Leonardi, P.M. When flexible routines meet flexible technologies: Affordance, constraint, and the imbrication of human and material agencies. *MIS Quarterly*, 35, 1 (2011), 147-167.

52. Leonardi, P.M., and Barley, S.R. Materiality and change: Challenges to building better theory about technology and organizing. *Information and Organization*, 18, 3 (2008), 159-176.

53. Lindberg, A., and Lyytinen, K. Towards a theory of affordance ecologies. In, Mitev, N., and de Vaujany, F.-X., (eds.), *Materiality and space*, Gordonsville, VA: Palgrave Macmillan, 2013, pp. 41-94.

54. Majchrzak, A., Faraj, S., Kane, G.C., and Azad, B. The contradictory influence of social media affordances on online communal knowledge sharing. *Journal of Computer-Mediated Communication*, 19, 1 (2013), 38-55.

55. Maxwell, J.A., and Mittapalli, K. Realism as a stance for mixed methods research. In, Tashakkori, A., and Teddlie, C., (eds.), *Sage handbook of mixed methods in social & behavioural research*, Thousand Oaks, CA: Sage, 2010, pp. 145-167.

56. Mell, P., and Grance, T. The nist definition of cloud computing. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Accessed Feb 27th, 2018.

57. Mingers, J., Mutch, A., and Willcocks, L. Introduction [special issue: Critical realism in information systems research]. *MIS Quarterly*, 37, 3 (2013), 795-802.

58. Olsson, H.H., and Bosch, J. *Climbing the "stairway to heaven": Evolving from agile development to continuous deployment of software.* Paper presented at the 38th Euromicro Conference on Software Engineering and Advanced Applications. Cesme, Izmir. 2014.

59. Orlikowski, W.J. Improvising organizational transformation over time: A situated change perspective. *Information Systems Research*, 7, 1 (1996), 63-92.

60. Orlikowski, W.J. Using technology and constituting structures: A practice lens for studying technology in organizations. *Organization Science*, 11, 4 (2000).

61. Pentland, B.T. Organizing moves in software support hot lines. *Administrative Science Quarterly*, 37, 4 (1992), 527-548.

62. Poppendieck, M., and Poppendieck, T. *Lean software development: An agile toolkit*. Reading, MA: Addison-Wesley, 2003.

63. Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L.E., Tiihonen, J., and Männistö, T. *Devops adoption benefits and challenges in practice: A case study.* Paper presented at the 17th International Conference on Product-Focused Software Process Improvement. Trondheim, Norway. 2016.

64. Sarker, S., Xiao, X., and Beaulieu, T. Guest editorial: Qualitative studies in information systems: A critical review and some guiding principles. *MIS Quarterly*, 37, 4 (2013), iii-xviii.

65. Schneider, S., and Sunyaev, A. Determinant factors of cloud-sourcing decisions: Reflecting on the it outsourcing literature in the era of cloud computing. *Journal of Information Technology*, 31, 1 (2016), 1-31.

66. Schwaber, K. *Agile project management with scrum*. Redmond, WA: Microsoft Press, 2004.

67. Stolberg, S. *Enabling agile testing through continuous integration.* Paper presented at the Agile Conference, 2009. AGILE'09. Chicago, IL. 2009.

68. Strong, D.M., Johnson, S.A., Tulu, B., Trudel, J., Volkoff, O., Pelletier, L.R., Bar-On, I., and Garber, L. A theory of organization-ehr affordance actualization. *Journal of the Association for Information Systems*, 15, 2 (2014), 53.

69. Susarla, A., Barua, A., and Whinston, A. A transaction cost perspective of the" software as a service" business model. *Journal of Management Information Systems*, 26, 2 (2009), 205-240.

70. Urquhart, C., Lehmann, H., and Myers, M.D. Putting the 'theory' back into grounded theory: Guidelines for grounded theory studies in information systems. *Information Systems Journal*, 20, 4 (2010), 357-381.

71. Venters, W., and Whitley, E.A. A critical review of cloud computing: Researching desires and realities. *Journal of Information Technology*, 27, 3 (2012), 179-197.

72. Vessey, I., and Conger, S. Learning to specify information requirements: The relationship between application and methodology. *Journal of Management Information Systems*, 10, 2 (1993), 177-201.

73. Vidgen, R., and Wang, X. Coevolving systems and the organization of agile software development. *Information Systems Research*, 20, 3 (2009), 355-376.

74. Volkoff, O., and Strong, D.M. Critical realism and affordances: Theorizing it-associated organizational change processes. *MIS Quarterly*, 37, 3 (2013), 819-834.

75. Walraven, S., Truyen, E., and Joosen, W. Comparing paas offerings in light of saas development. *Computing*, 96, 8 (2014), 669-724.

76. Walz, D.B., Elam, J.J., and Curtis, B. Inside a software design team: Knowledge acquisition, sharing, and integration. *Communications of the ACM*, 36, 10 (1993), 63-77.

77. Wettinger, J., Breitenbücher, U., and Leymann, F. *Devopslang–bridging the gap between development and operations.* Paper presented at the 3rd European Conference on Service-Oriented and Cloud Computing. Manchester, UK. 2014.

78. Wikibon. Platform-as-a-service market share worldwide in first half of 2015, by vendor. https://www.statista.com/statistics/478119/paas-vendor-market-share-ranking-worldwide/. Accessed Feb 27th, 2018.

79. Wikipedia. Feedback (disambiguation). https://en.wikipedia.org/w/index.php?title=Feedback_(disambiguation). Accessed Feb 27th, 2018.

80. Winkler, T.J., and Brown, C.V. Horizontal allocation of decision rights for on-premise applications and software-as-a-service. *Journal of Management Information Systems*, 30, 3 (2013), 13-48.

81. Yang, H., and Tate, M. A descriptive literature review and classification of cloud computing research. *Communications of the Association for Information Systems*, 31, 2 (2012), 35-60.

82. Zainuddin, E. *Secretly saas-ing: Stealth adoption of software-as-a-service from the embeddedness perspective.* Paper presented at the International Conference on Information Systems. Orlando, FL. 2012.